Received: 14 September 2020 | Accepted: 4 November 2020

DOI: 10.1002/cmm4.1143

SPECIAL ISSUE PAPER

WILEY

An improved C++ Poisson series processor with its applications

José Antonio López Ortí | **Vicente Agost Gómez** | **Miguel Barreda Rochera**

Department of Mathematics, Jaume I University of Castellón, Castellón, Spain

Correspondence
José Antonio López Ortí, Department of Mathematics, Jaume I University of Castellón, Castellón, Spain.
Email: lopez@mat.uji.es

Abstract

One of major problems in celestial mechanics is the management of the long developments in Fourier or Poisson series used to describe the perturbed motion in the planetary system. In this work we will develop a software package suitable for managing these objects. This package includes the ordinary arithmetic operations with Poisson series—such as sum or product—as well as the most commonly used functions sin, cos, or exp, among others. Derivation or integration procedures with respect to time of these objects have been implemented and inversion or serialization procedures are also attainable to obtain such series. All the programming has been accomplished overloading the arithmetic and functional operators natural to C++ with the intention of allowing the programmer to work in a more friendly way and treating the series as if they were mere numbers. In this work we extend the processor in order to obtain the solution to perturbed problems which solution is in the form of a Poisson series. These algorithms have been written in C++.

KEYWORDS

algorithms, celestial mechanics, computational algebra, orbital mechanics, perturbation theory, planetary theories

1 | INTRODUCTION

One of the main goals of celestial mechanics is to study the planetary motion in the solar system, that is, the construction of planetary theories. Planetary theories can be classified as analytical, semianalytical, and numerical. The analytical theories are based on the literal development of the perturbation function.[1-3] The semianalytical theories are based on developments of the perturbative potential using numerical values for the coefficients of Fourier series and literal expressions for the arguments of the trigonometric functions.[4,5] Numerical theories can be obtained by numerical integration of the motion equations. To this purpose it may be advisable to consult.[6]

To construct analytical and semianalytical theories it is necessary to obtain the developments of the main magnitudes of the two-body problem as Fourier series of the anomalies. To this purpose see References 2,3,7 if the mean anomalies are used and for other anomalies.[8-12]

The solution of the two-body problem is a well-known problem in celestial mechanics and its motion around the primary is given through of the elements $\vec{\sigma}$; it is quite common to use Brouwer and Clemence's set III nomenclature.[1] The perturbed problem can be studied by using the method of variation of constants, which is given by the Lagrange

 © 2020 John Wiley & Sons Ltd.

planetary equations

$$\begin{aligned}
\frac{da}{dt} &= \frac{2}{na}\frac{\partial R}{\partial \sigma},\\
\frac{de}{dt} &= -\frac{\sqrt{1-e^2}}{na^2 e}\frac{\partial R}{\partial \omega} + \frac{1-e^2}{na^2 e}\frac{\partial R}{\partial \sigma},\\
\frac{di}{dt} &= -\frac{1}{na^2\sqrt{1-e^2}\sin i}\frac{\partial R}{\partial \Omega} + \frac{\operatorname{ctg} i}{na^2\sqrt{1-e^2}}\frac{\partial R}{\partial \omega},\\
\frac{d\Omega}{dt} &= \frac{1}{na^2\sqrt{1-e^2}\sin i}\frac{\partial R}{\partial i},\\
\frac{d\omega}{dt} &= \frac{\sqrt{1-e^2}}{na^2 e}\frac{\partial R}{\partial e} - \frac{\cos i}{na^2\sqrt{1-e^2}\sin i}\frac{\partial R}{\partial i},\\
\frac{d\sigma}{dt} &= -\frac{2}{na}\frac{\partial R}{\partial a} - \frac{1-e^2}{na^2 e}\frac{\partial R}{\partial e},
\end{aligned} \tag{1}$$

where σ is defined by means of the equation:

$$M = \sigma + \int_{T_0}^{t} n\, dt, \tag{2}$$

and it coincides with ϵ in the case of the unperturbed motion. R is the disturbing potential, $R = \sum R_i$, due to the disturbing bodies $i = 1, \ldots, N$ and is defined as[7]

$$R = \sum_{k=1}^{N} G m_k \left[\left(\frac{1}{\Delta_k} \right) - \frac{x \cdot x_k + y \cdot y_k + z \cdot z_k}{r_k^3} \right], \tag{3}$$

where $\vec{r} = (x, y, z)$ and $\vec{r}_k = (x_k, y_k, z_k)$ are the coordinates of the secondary and the disturbing body with respect to the primary, Δ_k is the distance between the secondary and the disturbing body k, and m_k is the mass of the k body; finally, $\overrightarrow{\sigma_1}, \ldots, \overrightarrow{\sigma_N}$ are the elements of the N bodies.

In order to integrate the Lagrange planetary equations by analytical or semianalytical methods, in the first place, we solve the zero-order two-body problem Sun-planet for each planet. In the second place, we replace in the Lagrange planetary equations the solutions obtained in the zero-order problem. Thus, the solution in the $k+1$ order is obtained by replacing the k order solution in the second member of Lagrange planetary equations. The solution for the j-element of the i-planet of the planetary equations in order k is given by a development like

$$\sigma_{i,j} = \sum A t^i \cos(n_1 M_1 + \cdots + n_N M_N + B). \tag{4}$$

The terms of this series are the so-called Poisson terms and the complete series is known as a Poisson series. To integrate the Lagrange planetary equations it is necessary to work with these objects. The management of these series is quite difficult and it is convenient to have a Poisson series processor in order to manipulate these objects in a natural form.

A Poisson series processor is a software package containing a set of special functions programmed to deal with Poisson series. The first Poisson series processor was developed by Broucke[13] and it was coded in Fortran language. Other Poisson series processors—designed for specific purposes—were developed by Ivanova,[14] Abad,[15] and Navarro.[16] In the next section a new special software package to improve the handling of series of Poisson in the perturbative methods.

To construct a Poisson series processor it is necessary to define a previous C++ class called tPoisson, which members are the double precision numbers A, B, defining the amplitude and the phase of the Poisson term; the integer i, that represents the time exponent; and $n_1, \ldots, n_N$, which symbolizes the angular coefficients of the mean anomalies of the Poisson term.

This class contains methods to set and get the values of the members and methods, such as ==, >=, or !=, declared to compare two Poisson terms following specific criteria. The methods have been implemented using the operator overloading technique in order to get a comfortable usage for the final user.

Based on the previous concepts and on the native C++ vector class, a new class—named sPoisson—has been developed. The members of this new class are vectors of terms of Poisson. This class contains the methods of the class vector and a set of private methods to sort, compact, or arrange Poisson series.

The main public methods included in poisson.h are the arithmetic operations +, −, *, *pow*; the extension of the most common functions sin, cos, exp … to be evaluated over Poisson series;[17] and functional operations such as Taylor developments and series inversion procedures based on Lagrange and Deprit methods.[18] The operators and common functions have been overloaded in order to be more user friendly.

In order to guarantee a unified representation of the Poisson series, the following rules are to be followed:

1. If at least one k_j coefficient is not zero, the first not zero coefficient will be positive, A will be positive, and $B \in [0, 2\pi]$. If all k_j are zero, then A will be replaced by $A \cdot \cos B$ and B will be set to zero.
2. In a Poisson series only one term with the same coefficients $k_1, \ldots, k_p$ and the same powers of the spatial variables can be held. If recurrence occurs, the processor will merge the similar terms in the appropriate way.

In this work a processor dealing with vectors of Poisson series has been built. A Poisson series vector, *vPoisson* $sp[N+1]$—with N fixed in advance is an object where $sp[0], sp[1], \ldots, sp[N]$ respectively represent the terms of order $0, 1, \ldots, N$ in ε of the solution of a perturbed problem, which its solution can be developed as a power series of ε. This processor evaluate in an automatic form the development of the second members of perturbed equation, thus avoiding having to develop in Taylor series in manual mode, which is generally very tedious.

In Section 1 we will introduce the problem by defining the Poisson series and its usage problems; this section contains the general background of the problem.

In Section 2, the Poisson series processor is extended in order to manage these objects as vectors $sp[N+1]$ containing $sp[k]$ $k=0, \ldots, N$ a Poisson series containing the terms of order k in a parameter ε. This section encloses a subsection explaining the content of the main classes of the taylor_poisson.h package. In Section 3, a set of simple numerical examples are developed in order to show the performance of the new processor. In Section 4, the main conclusions of this work are exposed.

2 | THE TAYLOR PROCESSOR SERIES

In these paragraphs we introduce the problems depending on one or more small paramaters $\varepsilon_1, \ldots, \varepsilon_k$, which orders are $o^1, \ldots, o^k$. In this article we will only focus on one-parameter problems, as its explanation is simpler and the generalization to more parameters can be easily accomplished by taking the greater as ε and the rest as $\alpha_i\varepsilon$. The method used in perturbation theory to solve the problem $\dot{\vec{x}} = \vec{f}(\vec{x}, \varepsilon, t)$ consists of the preliminary integration of the differential equations for $\varepsilon = 0$. We obtain a solution $\vec{x} = \vec{x}(\vec{A}, t)$, where $\vec{A}$ is a set of constants equaling the number of variables which values are determined by the initial conditions of the system.

A commonly used technique to solve the whole problem ($\varepsilon \neq 0$) is the method known as variation of constants, which swaps the constants $A_1, \ldots, A_n$ with the functions $A_1(t), \ldots, A_n(t)$. Thus, the substitution of these functions in $\vec{x}(\vec{A}, t)$ satisfies the complete equation. The method of variation of constants in many problems leads to linear differential equations of the form $\frac{d\vec{A}}{dt} = F(A_1(t), \ldots, A_n(t), t) \cdot \vec{A}(t)$, where F is a matrix, which elements have the same order as ε. Generally, the method used to solve this system is the method of successive iterations. Starting with the values of $\vec{A}$ that appear in the unperturbed problem we obtain a solution $\delta_1\vec{A}(t)$ of first order in ε. Coming out of this solution, $\vec{A}(t) + \delta_1\vec{A}(t)$, we develop in second order in $\delta_1\vec{A}(t)$ and we obtain $\frac{d\delta_2\vec{A}}{dt} = D(F(\vec{A}(t), t))\delta_1\vec{A}(t)$ and the process goes on successively.

In many problems, such as planetary theories, mathematical perturbed pendulums or with large oscillations, the quantities $\delta_r\vec{A}(t)$ adopt the form of a Poisson series, so, we are aimed to extend our processor in order to be able to automatically calculate the values of $\frac{d\delta_2\vec{A}}{dt}$, $\frac{d\delta_3\vec{A}}{dt}$, and so on up to order N, established in advance.

The method is simple and it is based on the representation of each $A_i(t)$ with a Poisson series vector of dimension N, where $A[0](t)$ represents zero-order perturbation terms, $A[1](t)$ the first-order ones, and up to $A[N](t)$, that designates the terms of order N in ε. With these vectors it is easy to provide an algebra substantiating

$$sp(A[0], A[1], \ldots, A[n]) = (spA[0], spA[1], \ldots, spA[n]), \tag{5}$$

defining the scalar product,

$$(A[0], A[1], \ldots, A[N]) + (B[0], B[1], \ldots, B[n]) = (A[0] + B[0], A[1] + B[1], \ldots, A[N] + B[N]), \tag{6}$$

determining the addition of two Poisson series vectors, and finally

$$(A[0], A[1], \ldots, A[N]) \cdot (B[0], B[1], \ldots, B[n]) = (C[0], C[1], \ldots, C[N]), \quad C[k] = \sum_{i=0}^{k} A[i]B[k-i], \tag{7}$$

that defines the product of two Poisson series vectors. In the previous equations $sp, A[0], \ldots, A[N], B[0], \ldots, B[N]$ represent Poisson series and it is noticeable that the operations maintain the order in ε of the components of the vector in all cases.

With these operations it is possible to manipulate Poisson series of the form $\sum_{i=0}^{N} sp[i]\varepsilon^i$, represented as vectors $(sp[0], sp[1], \ldots, sp[i])$, in a way that allows to redefine all the operations and functions, such as $\sin(sp)$, $\cos(sp)$, $\exp(sp)$, $\log(1+sp)$, $(1+sp)^r$, and so on, included in the Poisson series processor posison_vector.h. Functions and operators can also be overloaded and this allows to create another C++ class, namely taylor_poisson.h. This new class empowers a more friendly and natural utilization of the equations linked to the method of variation of constants in the perturbed problem. Besides, the writing of the code dealing with that problem in different orders of perturbation can be accomplish more easily. This prevents the programmer from manually developing the Taylor series of the right-hand side of the equations, which in cases such as the Lagrange equations can be extraordinarily tedious.

The Poisson series processor on which the one presented here is based has been extensively tested, as a sample in the case of an analytical planetary theory.[19] The kernel of the class vPoisson is schematically shown as follows.

2.1 | Classes tPoisson and sPoisson

First, we lightly describe the classes previously built: tPoisson, a Poisson term; and sPoisson, a Poisson series. We will then define the new class vPoisson, a vector Poisson series, based on the mentioned classes before.

Class tPoisson

Hence, the tPoisson class is shown in more detail. The class harbors a data structure to be capable of storing mathematical expressions of the form

$$Tp = A \cdot t^i \cdot \cos\left(\sum_{k=1}^{NP} n_k \cdot b_k + B\right), \tag{8}$$

which will be known as Poisson terms, and a set of specific methods.

This class will be the fundamental tool to be able to create, manipulate and, ultimately, operate Poisson terms and construct Poisson series as a vector of Poisson terms.

Members:

- *m_A*. A double precision value to store the variable A.
- *m_B*. A double precision value to store the variable B.
- *m_i*. A whole value to store the exponent i.
- *m_j*. A static vector of whole values to store the NP variables $n_1, \ldots, n_k$.

Methods:

- *tPoisson()*. A constructor to create an instance of Poisson term with all its values set to zero.
- *tPoisson (const tPoisson&)*. A copy constructor to create a Poisson term; the new term is copied from a given one.
- *tPoisson (double, int, int[], double)*. A constructor to create a Poisson term explicitly specifying all its members.

- *tPoisson (double)*. A constructor to create a Poisson term by defining solely the member m_A and nullifying the rest. This method will let us create a Poisson term associated to a real number.
- *tPoisson()*. An instance destructor to delete Poisson terms and free the memory consumed.
- *double getA()*. Method to access and read the value of the member m_A.
- *void setA(const double&)*. Method to allocate a value to the member m_A in a term.
- *double getB()*. Method to retrieve the value of member m_B.
- *void setB(const double&)*. Method to allocate a value to the member m_B of a term.
- *int getI()*. Method to retrieve the value of member m_I.
- *void setI(const int&)*. Method to allocate a value to the member m_I of a term.
- *int getJ(int)*. This method retrieves the value of the angular variable of a term given its position in the vector.
- *void setJ(int, int)*. Method to allocate the value of an angular variable given its position in the vector.

Operators:

- *bool operator== (const tPoisson&, const tPoisson&)*. This function overloads the == operator in order to be able to compare two Poisson terms. For the proper functioning of the processor it is important to highlight that two Poisson terms are considered equal if both their members, m_I, and their vectors, m_J, coincide.
- *bool operator!= (const tPoisson&, const tPoisson&)*. Overloads the != operator in order to be able to determine if two Poisson terms are different. Two terms are different if there are unequal. The definition of the function for this operator has been constructed based on the negation of the equality operator.
- *bool operator> (const tPoisson&, const tPoisson&)*. Overloads the > operator allowing to decide, under certain criteria, if a Poisson term is greater than another.
- *bool operator>= (const tPoisson&, const tPoisson&)*. Both the definitions of the overloaded functions > and == have been defined; the operator >= combines both of them.
- *bool operator< (const tPoisson&, const tPoisson&)*. Overloads the < operator allowing to decide, under certain criteria, if a Poisson term is less than another.
- *bool operator<= (const tPoisson&, const tPoisson&)*. The <= operator is a combination of the functions < and == and both operators have been previously implemented.

Class sPoisson

The class sPoisson is a data structure able to store a mathematical expression of the form

$$\sum_{i=1}^{M} Tp_i \quad \text{where } Tp_i \text{ is a Poisson term,}$$

which will be known as a Poisson series. It is, indeed, a vector whose elements are Poisson terms. To implement the data structure the C++ standard library has been exploited and the C++ vector container has been taken to support the recently created sPoisson class defined as vector<tPoisson>

Members:

The members of the class sPoisson are the same ones as the C++ standard library container vector, since it has been defined as #sPoisson vector<tPoisson>.

Methods:

The methods of this class are the methods corresponding to the vector class and the ones inherited from the tPoisson class. Moreover, some operators have been overloaded and some functions have also been programmed.

Operators:

- *sPoisson& operator*(sPoisson&, sPoisson&)*. This operator has been devised to multiply two Poisson series.

- *sPoisson& operator*(const sPoisson&, double)*. This operator right-multiplies a Poisson series by a real number.
- *sPoisson& operator*(double, const sPoisson&)*. In the case, this operator left-multiplies a Poisson series by a real number.
- *sPoisson& operator/ (const sPoisson &, double)*. Operator to divide a Poisson series by a real number.
- *sPoisson& operator+(const sPoisson&, const sPoisson&)*. Operator to perform the addition of two Poisson series.
- *sPoisson& operator+(const sPoisson&, double)*. Operator to right-add a Poisson series and a real number.
- *sPoisson& operator+(double, const sPoisson&)*. Operator to left-add a Poisson series with a real number.
- *sPoisson& operator-(const sPoisson&, const sPoisson&)*. Operator to subtract a Poisson series from another.

Other functions:

In addition to the members of each previous classes, some manipulative functions involving Poisson series have been programmed. All these functions depend on the basic operations defined by the operators of the class.

- *sPoisson& pow(sPoisson&, int)*. This function has two parameters: a Poisson series and whole number representing the exponent. Based upon the operator "*," this function implements the power of a Poisson series by multiplying the series by itself as many times as the value indicated by the exponent. It returns the compacted Poisson series after all the operations have been performed.
- *sPoisson& exp(sPoisson&)*. This function admits a Poisson series as a parameter. According to the established precision, it computes an exponential approximation based on Taylor's formula and returns a new compacted Poisson series.
- *sPoisson& sin(sPoisson&)*. This function admits a Poisson series as a parameter. According to the established precision, it computes an approximation of the function sine based on the Taylor's formula and returns a new compacted Poisson series.
- *sPoisson& cos(sPoisson&)*. This function admits a Poisson series as a parameter. According to the established precision, it computes an approximation of the function cosine based on the Taylor's formula and returns a new compacted Poisson series.
- *sPoisson& sqrt(sPoisson& sp, double)*. This function admits two parameters: a Poisson series and the index of the root. According to the established precision, it computes an approximation of the function $(1+x)^{\frac{1}{p}}$ based on the Taylor's formula and returns a new compacted Poisson series.
- *sPoisson& log(sPoisson&)*. This function admits a Poisson series as a parameter. According to the established precision, it computes an approximation of the function $\log(1+x)$ based on the Taylor's formula and returns a new compacted Poisson series.
- *void arregla(sPoisson&)*. This function alters the content of a Poisson series in order to meet the convenient characteristics for its future treatment. If all the angular variables (m_j) are zero and m_B is not, it is advisable to make cos(m_B) part of m_A. Besides, it is also desirable to reduce the angles to the interval $[0, 2\pi[$.
- *sPoisson& trunca(const sPoisson&, double)*. This function accepts a Poisson series and a positive number representing the maximum error as parameters. It returns another Poisson series where the terms with a smaller amplitude than the maximum error have been deleted.
- *sPoisson& compacta(const sPoisson&)*. It accepts a Poisson series as a parameter and computes another series where its terms have been arranged, operated, unified and simplified (equality holds depending on the overloaded operator == defined in the class tPoisson).

2.2 | Class vPoisson

Based on the previous classes—tPoisson and sPoisson—and their basic operations defined in (5), (6), and (7) we can build the vPoisson class. This new class is the core that will allow the construction of the Taylor series processor.

Members:

The members of this class are the members of the vector class included in the C++ standard library, as the class has been implemented by means of #vPoisson vector<sPoisson>.

Operators:

- *vPoisson& operator+(const vPoisson&, const vPoisson&)*. This operator admits two vectors of Poisson series as parameters. It computes the addition of both vectors and returns the result as another vector.
- *vPoisson& operator*(const vPoisson&, const vPoisson&)*. This operator admits two vectors of Poisson series as parameters. It computes the product of both series. Especially in this operator it is important to highlight that the order in the series is always held, as terms of the same order must be kept together.

Other functions:

- *vPoisson& pow(const vPoisson&, int)*. This function admits two parameters: a vector and an positive integer. It multiplies the vector as times as the integer indicates, but always maintaining the terms of the same order in its adequate position within the series in the vector.
- *vPoisson& sin(const vPoisson&)*. This function admits a Poisson series vector as a parameter. According to the established precision, it computes an approximation of the sine function based on the Taylor's formula and returns a vector of series. Within this new vector the terms of the same order are kept together as part of the same Poisson series.
- *vPoisson& cos(const vPoisson&)*. This function admits a Poisson series vector as a parameter. According to the established precision, it computes an approximation of the cosine function based on the Taylor's formula and returns a vector of series. Within this new vector the terms of the same order are kept together as part of the same Poisson series.
- *vPoisson& exp(const vPoisson&)*. This function admits a Poisson series vector as a parameter. According to the established precision, it computes an approximation of the exponential function based on the Taylor's formula and returns a vector of series. Within this new vector the terms of the same order are kept together as part of the same Poisson series.
- *vPoisson& log(const vPoisson&)*. This function admits a Poisson series vector as a parameter. According to the established precision, it computes an approximation of the $\log(1+x)$ function based on the Taylor's formula and returns a vector of series. Within this new vector the terms of the same order are kept together as part of the same Poisson series.
- *vPoisson& compacta(vPoisson&)*. This function admits a Poisson series vector as a parameter. As the vector comprises a list of several Poisson series, it individually manipulates each one by applying the function compacta, previously defined within the class sPoisson.

By way of example, next section includes several problems in order to prove the efficiency of the package.
The kernel including the classes introduced before can be found in the URL http://mecanicaceleste.uji.es.

3 | NUMERICAL EXAMPLE

To test the efficiency of our processor we will solve a simple problem via perturbative methods: a mathematical pendulum. It is defined by

$$\frac{d^2x}{dt^2} = -\omega^2 \sin x, \quad x(0) = 0.5, \quad \dot{x}(0) = 0. \tag{9}$$

The exact solution to this problem is a well-known result[20] and is given in terms of Jacobi's elliptical functions;[20,21] the period is smaller than the one used in the approximation

$$\frac{d^2x}{dt^2} = -\omega^2 x,$$

commonly used for small values of the amplitude and also suitable for the zero-order approximation in our problem.

The exact solution of the mathematical pendulum $\ddot{x} = -\omega^2 \sin x$ is given by[20]

$$\sin\left(\frac{x}{2}\right) = \sin\frac{\alpha}{2}\mathrm{sn}\,(\omega t, k), \tag{10}$$

where α is the amplitude of the motion, and sn the elliptic sinus of Jacobi for $k = \sin\frac{\alpha}{2}$ defined as

$$u = \int_0^{\phi} \frac{d\phi}{\sqrt{1 - k^2 \sin^2\phi}}, \tag{11}$$

where $\phi = \text{am}(u)$ and $\text{sn}\ u = \sin(\text{am}\ (u)) = \sin\phi$.

The period of this motion is given by $T = \frac{4K(k)}{\omega}$ where $K(k)$ is the complete elliptic integral of the first kind

$$K(k) = \int_0^{\pi/2} \frac{d\varphi}{\sqrt{1 - k^2 \sin^2\varphi}}. \tag{12}$$

Notice that $\cos\frac{x}{2} = \sqrt{1 - k^2 \text{sn}\ (\omega t, k)} = \text{dn}\ (\omega t, k)$ and so $\dot{x} = 2\omega \sin\frac{\alpha}{2} \text{cn}\ (\omega t, k)$.

In order to simplify the calculations, temporal units have been chosen conducive to have $T = 2\pi$ for the linear approximation, and so, $\omega = 1$.

If the amplitude is not small, the model of harmonic oscillator cannot be applied to mathematical pendulum. For example, for $A = 0.9$, we have that the period of linear oscillator is $T = 2\pi = 6.28319$ and the real period is $T_r = 6.61687$.

To determine the solution to our problem, firstly, we consider the development of the series $\sin x$ and, secondly, we introduce the parameter ε. The parameter is chosen so that if $\varepsilon = 0$, we have a linear approximation and if $\varepsilon = 1$, the problem is complete. Therefore, we study the problem

$$\frac{d^2x(t)}{dt^2} = -x + \sum_{k=1}^{\infty} (-1)^{k+1} \varepsilon^k \frac{x^{2k+1}}{(2k+1)!}, \tag{13}$$

with the initial conditions $x(0) = 0.5$, $\dot{x}(0) = 0$.

According to the previous transformations we have changed the problem into one eligible to be approximated through perturbation theory; the solution would be of the form

$$x(t) = \sum_{k=0}^{\infty} \varepsilon^k x_k(t). \tag{14}$$

Assuming zero-order, this problem turns into a harmonic oscillator, which solution is given by

$$x_0(t) = A\cos t + B\sin t, \tag{15}$$

and considering the initial conditions, we have $A = 0.5$, $B = 0$. If first-order in ε terms are taken, we attain the classical problem of the Duffing oscillator.[20]

The Duffing oscillator problem

$$\ddot{x} = -x + \mu x^3, \tag{16}$$

associated to mathematical pendulum can be solved by using a perturbative method by means of the introduction of a parameter ε.

$$\ddot{x} - x + \varepsilon \frac{x^3}{6}. \tag{17}$$

The solution of unperturbed problem is known and applying the constant variation method we obtain:

$$\begin{aligned} \dot{A}(t) &= -\sin(t)\ \frac{x^3}{6}\varepsilon, \\ \dot{B}(t) &= \quad \cos(t)\ \frac{x^3}{6}\varepsilon. \end{aligned} \tag{18}$$

The terms in first order in ε are

$$x1[t] = 0.0078125t\sin(t) + 0.000651042\cos(t) - 0.000651042\cos(3t), \tag{19}$$

with period $T_1 = 6.38237$.

The second approximation in ε is

$$x2[t] = (-6.104t^2\cos(t) + 8.138t\sin(t) - 3.052t\sin(3t) + 1.950\cos(t) - 2.035\cos(3t)) \cdot 10^{-5}, \tag{20}$$

with period $T_2 = 6.38383$.

The terms of third order in ε are

$$x3[t] = (-1.19t^2\cos(t) + 1.83t\sin(t) - 1.23t\sin(3t)) \cdot 10^{-6}, \tag{21}$$

with period $T_3 = 6.38383$.

The Duffing oscillator improves the solution given by linear oscillator; however, if the initial amplitudes are large, the periods do not converge to the true period of the mathematical pendulum.

To avoid this inconvenience we study the complete problem through perturbative methods where it is advisable to use Lagrange's method of variation of parameters. Thus, A and B are replaced in the solution to the unperturbed problem with $A(t)$, $B(t)$, satisfying

$$\begin{aligned} \dot{A}(t) &= -\sin(t)\sum_{k=1}^{\infty}(-1)^{k+1}\varepsilon^k\frac{x^{2k+1}}{(2k+1)!}, \\ \dot{B}(t) &= \cos(t)\sum_{k=1}^{\infty}(-1)^{k+1}\varepsilon^k\frac{x^{2k+1}}{(2k+1)!}. \end{aligned} \tag{22}$$

A classic solution using the perturbation method is as follows: to solve this system we apply a successive iterative technique. First, we will obtain the terms $A(t)$ and $B(t)$ as a first-order approximation in ε; these results are provided by the Poisson series processor. Then, after integrating and considering the initial conditions we obtain the terms up to third order of perturbation. They are shown below.

In order to show the iterations of this problem—and considering that t is the only angular variable—the terms $\cos(nt + \Psi)$ will be presented as $\cos(\Psi)\cos(nt) - \sin(\Psi)\sin(nt)$ in favor of greater clarity.

The unperturbed problem entails the solution

$$x0[t] = 0.5\cos(t), \tag{23}$$

with period $T_0 = 2\pi = 6.28319$.

After running the Poisson series processor, taking the first-order terms of the vectors of the series that we obtain from (33), integrating, and imposing the initial conditions we obtain

$$x1[t] = (0.651\cos(t) - 0.651\cos(3t) + 7.813t\sin(t))10^{-3}, \tag{24}$$

with period $T_1 = 6.38237$—which results greater than the harmonic oscillator. The results are truncated at 10^{-6}.

Another point worth underlining is the presence of a term in the form $t\ \sin(t)$, which is a nonperiodic term—in fact t derives from the Taylor approximations of periodic terms—and the approximation is only valid for not too large values of t. As the movement is periodic and symmetric with respect to $x = 0$, it is enough to calculate the solution up to $\frac{T}{4}$ and extend it by periodicity and symmetry.

The terms we get in second-order perturbation in ε are obtained by replacing in (33). Considering the initial conditions we have:

$$\begin{aligned} x2[t] = (&1.22t\cos(t) + 0.61t\cos(3t) + 0.31t\cos(5t) - 2.74\sin(t) \\ &+ 0.61t^2\sin(t) + 0.05\sin(3t) + 0.05\sin(5t) + 0.03\sin(7t))10^{-4}, \end{aligned} \tag{25}$$

being the period in this case $T_2 = 6.38136$.

Third-order perturbation follows the same procedure; we obtain:

$$x3[t] = \left((4.t + 1.t^2)\cos(t) + 2.t\cos(3t) + 3.t\sin(t) - (1. - 1.t^2\sin(3t))\right)10^{-6}. \tag{26}$$

The period in this final case results $T_3 = 6.38137$ and it remains unchanged in a fourth-order perturbation.

The solution to the problem up to third-order perturbation is given by

$$x[t] = x0[t] + x1[t] + x2[t] + x3[t]. \tag{27}$$

The periods are computed by solving the equation $xi(t) = 0$ around the point $t_0 = \pi/4$ and considering the root ξ_1, we have $T_i = 4\xi_i$ which slightly differs from the real one—the one provided by elliptic functions—in less than $4 \cdot 10^{-6}$ in the interval $[0, T_3/4]$.

In this case, the problem can be written as

$$\ddot{x} = -x + x - \sin x, \tag{28}$$

and in this form we can consider the problem as an harmonic oscillator perturbed by a force $F = x - \sin x$. In the method applied in the previous section it is easy to develop the perturbative force in Taylor series as

$$x - \sin x = -\sum_{k=1}^{\infty} (-1)^k x^{2k+1}/(2k+1)!, \tag{29}$$

and from this development we can introduce the problem depending on a parameter ε

$$\ddot{x} = -x - \sum_{k=1}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} \varepsilon^k. \tag{30}$$

This problem is solved by using the perturbation theory arranging an appropriate precision for one-fourth of the period, and from this solution we can extend them for other time.

Generally, the development of the perturbative forces is not easy, for example, in (1) and in this case it is preferable to leave the algebra to be carried out by a proper Poisson processor series. In this sense, in our example it is convenient to study the problem:

$$\ddot{x} = -x + \varepsilon(x - \sin x). \tag{31}$$

Our original problem is satisfied when $\varepsilon = 1$ and the method to solve this problem is to approach the solution of the perturbed problem as

$$x(t) = x0(t) + x1(t)\varepsilon + x2(t)\varepsilon^2 + \ \dots \ , \tag{32}$$

and from this solution and taking $\varepsilon = 1$ we obtain an approximation to the mathematical pendulum solution.

The general solution of unperturbed problem is given by (15) and in this case the equations given by the method of variation of constants are

$$\begin{aligned} \dot{A}(t) &= -\sin(t)\ (x - \sin x)\varepsilon, \\ \dot{B}(t) &= \quad \cos(t)\ (x - \sin x)\varepsilon. \end{aligned} \tag{33}$$

Applying the Taylor series processor to this problem we obtain in first order in ε

$$x1(t) = (0.64264\cos(t) - 0.64093\cos(3t) + 0.00067\cos(5t) + 7.73154t\sin(t))10^{-3}, \tag{34}$$

with period $T_1 = 6.38134$

In second order in ε we have

$$x2(t) = (1.981\cos(t) - 1.98085\cos(3t) + 0.084\cos(5t) + 7.952t\sin(t) - 0.029t\sin(3t) + 0.005\sin(5t) - 5.978t^2\cos(t))10^{-5}, \tag{35}$$

with period $T_2 = 6.38276$

Finally, in third order in ε we get

$$x3(t) = (0.58\cos(t) - 0.63 - 0.63\cos(3t) + 0.05\cos(5t) + 1.76t\sin(t) - 1.115t^2\cos(t) + 0.69t^2\cos(37) - 0.31t^3\sin(t))10^{-6}, \tag{36}$$

with period $T_3 = 6.38279$

To sum up, the example in this section demonstrates the precision of the method. The solution of the mathematical pendulum with initial conditions $x(0) = 0.5$, $\dot{x}(0) = 0$ is obtained and the periods for each order of perturbation are shown.

4 | CONCLUDING REMARKS

The usage of Poisson series is of utmost importance in the performance of perturbative methods in celestial mechanics. In this article the processor included in the class poisson.h has been extended. This extension has been successfully tested in problems as complex as the development of first-order planetary theories. Using the processor in higher-order theories involves the development of the second members of the Lagrange planetary equations as Taylor series, which is a burdensome process.

The taylor_poisson.h processor deals with vPoisson objects defined as Poisson series vectors. The components of these vectors correspond to the order of the small parameter around which the perturbative method is employed. The potential advantage of this processor is that the manual development in Taylor series of the small parameter ε is not required, as the processor has been programmed to do all the work automatically.

The efficiency of the processor has been tested in a simple example involving a mathematical pendulum with nonsmall oscillations, where linear modeling is not appropriate. On the one hand, a clean illustrative example has been chosen because the prime objective is to focus on the simplicity of usage of the processor when dealing with different perturbative orders; on the other hand, we want to compare the results with the ones produced by Mathematica, which is a unviable task when dealing with problems involving planetary theories.

As summary, we can highlight that

- The Poisson series processor is suitable for the ordinary requirements of celestial mechanics.
- The Poisson series vector processor is appropriate to study, by the method of variation of parameters, problems close to problems admitting periodic solutions.
- The exemplification of the solution for the oscillations of a mathematical pendulum using a perturbative method when the amplitude is small can be handily automated with a Poisson series vector processor.
- The results obtained with the processor—which kernel is included in the class taylor_poisson.h—have been compared with the ones achieved with Mathematica and they are exactly the same.
- The method followed is extensible to more complicated problems, such as the analytical theory of an artificial satellite, planetary theories, and other problems linked with celestial mechanics.
- The kernel of the processor is included in the C++ class taylor_poisson.h and can be accessed under General Public License v3 at the URL http://mecanicaceleste.uji.es.

ACKNOWLEDGMENTS

This information is not included in this version in order to do not show information about the authors.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest with regard to this work.

ORCID

José Antonio López Ortí https://orcid.org/0000-0002-9620-6454

REFERENCES

1. Brower D, Clemence GM. *Celestial Mechanics*. New York, NY: Academic Press; 1965.
2. Hagihara Y. *Celestial Mechanics*. Cambridge MA: MIT Press; 1970.
3. Tisserand FF. *Traité de Mecanique Celeste*. Paris, France: Gauthier-Villars; 1896.
4. Chapront J, Bretagnon P, Mehl M. Un formulaire pour le calcul des perturbations d'ordres élevés dans les problèmes planétaires. *Celest Mech*. 1975;11:379-399.
5. Kovalevsky J. *Introduction to Celestial Mechanics*. Dordrecht-Holland: D. Reidel Publishing Company; 1967.

6. Vigo-Aguiar J, Ferrandiz JM. A general procedure for the adaptation of multistep algorithms to the integration of oscillatory problems. *Siam J Numer Anal*. 1998;35:1684-1708.
7. Levallois LL, Kovalewsky J. *Geodesie Generale*. Vol 4. Paris, France: Eyrolles; 1971.
8. Chapront J, Simon JL. Planetary theories with the aid of the expansions of elliptic functions. *Celest Mech*. 1996;63:171-188.
9. Brumberg VA, Brumberg EV. Elliptic anomaly in constructing long-terms and short-term dynamical theories. *Celest Mech*. 2001;80:159-166.
10. Brumberg VA, Fufkushima T. Expansions of elliptic motion based on elliptic functions theory. *Celest Mech*. 1994;60:1-36.
11. López JA, Barreda M. A formulation to obtain semi-analytical planetary theories using true anomalies as temporal variables. *J Comput Appl Math*. 2007;2004:77-83.
12. López JA, Martínez MJ, Marco FJ. Semi-analytical integration algorithms based on the use of several kinds of anomalies as temporal variable. *Planet Space Sci*. 2008;32:191-198.
13. Broucke R, Garthwaite K. A programming system for analytical series expansions on a computer. *Celest Mech*. 1969;1:271-284.
14. Ivanova T. A new echeloned Poison series processor. *Celest Mech*. 2001;80:167-176.
15. Abad A, San-Juan JF. PSPCLink: a cooperation between general symbolic and Poisson series processors. *J Symbol Comput*. 1997;24:113-122.
16. Navarro JF, Ferrandiz JM. A new symbolic processor for the earth rotation. *Celest Mech*. 2002;82:243-263.
17. Brumberg VA. *Analytical Techniques of Celestial Mechanics*. Berlin, Germany: Springer-Verlag; 1995.
18. Deprit A. Note on Lagrange's inversion formula. *Celest Mech*. 1979;20:325-327.
19. López JA, Agost V, Barreda M. An improved algorithm to develop semi-analytical planetary theories using Sundman generalized variables. *J Comput Appl Math*. 2015;275:403-411.
20. Lawden Elliptic DF. *Functions and Applications*. Beijing, China: College Press, University of Beijing; 1989.
21. Lavrentev MA, Shabat BV. *Methods of the Theory of Function of Complex Variable*. Moscow, Russia: Nauka; 1987.

AUTHOR BIOGRAPHIES

José Antonio López Ortí Academic background: (1) Master in Mathematics science (1982). (2) Doctor in Mathematics (1987). He was Assistant since 1983 to 1984, Lecturer since 1984 from 1989, Professor (associated) since 1989, and Full Professor since 2011 to now. https://www.uji.es/departaments/mat/base/estructura/personal/?urlRedirect=https://www.uji.es/departaments/mat/base/estructura/personal/&url=/departaments/mat/base/estructura/personal&p_departamento=92&p_profesor=65082; https://www.researchgate.net/profile/Jose_Lopez_Orti

Vicente Agost Gómez—born in 1974 in Castellón, Spain—graduated in Computer Engineering in 1997 and in specific Computational Mathematics in 2008 at Jaume I. He holds a master degree in Computational Mathematics and earned a Ph.D. in Sciences at Jaume I university in 2020. He obtained his permanent position as a secondary school mathematics teacher in 2008 and joined Jaume I university as a part-time associate professor in the same year. He is presently conducting research into celestial mechanics and specifically into orbital motion.

Miguel Barreda Rochera was born in Castelló, Spain, in 1963. He went to primary school at the school Colegio Nacional de Prácticas Aneja in Castelló and to secondary school at the high school Francisco Ribalta in Castelló. He studies Mathematics at the University of Valéncia and received the Doctor degree in Mathematical Sciences from the University of Valéncia in 1994. From 1988 to 1990 he was Research Fellow at the Collegi Universitari de Castelló (University of Valéncia), Assistant Professor at University of Valéncia in 1991 and, since 1991 he is Associate Professor at the University Jaume I of Castelló. The focus of Dr. Barreda Rochera's research has been the study of rigid motions in General Relativity and Celestial Mechanics. ORCID: https://orcid.org/0000-0001-6876-5411

How to cite this article: López Ortí JA, Agost Gómez V, Barreda Rochera M. An improved C++ Poisson series processor with its applications. *Comp and Math Methods*. 2020;e1143. https://doi.org/10.1002/cmm4.1143

APPENDIX C++ CODE

This appendix contains the C++ code used to compute the solution of the problem (31) with initial conditions $x(0) = 0.5$, $\dot{x} = 0$ up to third order of perturbation. To start the solution of the unperturbed problem has been used.

```
#include "vpoisson.h"
sPoisson& integrate(const sPoisson& sp1);
main()
{
   // INITIALIZATION OF THE VARIABLES
   int n[1],k,T_order=3;
   n[0]=1;
   sPoisson aux0,aux1;
   aux0.push_back(tPoisson(0.));
   vPoisson x(4,aux0),xp(4,aux0),A(4,aux0),B(4,aux0),AP(4,aux0),BP(4,aux0),vst(4,aux0),vct(4,aux0),vaux(4,aux0);
   sPoisson aux;
   double x0=0.5,xp0=0;
   long double M[1],A0[4],B0[4];
   M[0]=0;
   aux.clear();
   aux.push_back(tPoisson(0.5,0,n,0.));
   x[0]=aux;
   aux.clear();
   aux.push_back(tPoisson(0.5,0,n,PI_L/2));
   xp[0]=aux;
   B[0]=aux0;
   aux0.clear();
   aux0.push_back(tPoisson(0.5));
   A[0]=aux0;
   aux0.clear();
   aux0.push_back(tPoisson(1,0,n,3*PI_L/2));
   vst[0]=aux0;
   aux0.clear();
   aux0.push_back(tPoisson(1,0,n,0));
   vct[0]=aux0;
   // COMPUTATION OF THE PERTURBATED TERMS
   for(k=1;k<=T_order;k++)
   {
        vaux=sin(x);
        vaux=x-vaux;
        AP[k]=-vst[0]*vaux[k-1];
        A[k]=integrate(AP[k]);
        A0[k]=-(valor(A[k],0.,M));
        A[k]=A[k]+A0[k];
        BP[k]=vct[0]*vaux[k-1];
        B[k]=integrate(BP[k]);
        B0[k]=-(valor(B[k],0.,M));
        B[k]=B[k]+B0[k];
        x[k]=A[k]*vct[0]+B[k]*vst[0];
        printf("terms of %2i order in epsilon\n",k);
        show_sincos(x[k]);
   }
}
```

```
sPoisson& integrate(const sPoisson& sp1)
{
	int i,N=sp1.size(),n[1],m;
	long double A,B;
	sPoisson spint,sprest;
	tPoisson tp;
	for(i=0;i<N;i++)
	{
		tp=sp1[i];
		A=tp.getA();
		m=tp.getI();
		n[0]=tp.getJ(0);
		B=tp.getB();
		if(n[0]==0)
		{
			spint.push_back(tPoisson(A/(m+1),m+1,n,0));
		}
		else if(m==0)
		{
			spint.push_back(tPoisson(A/n[0],0,n,B+1.5*PI_L));
		}
		else
		{
			spint.push_back(tPoisson(A/n[0],m,n,B+1.5*PI_L));
			sprest.push_back(tPoisson(A/n[0]*m,m-1,n,B+0.5*PI_L));
		}	}
	if(sprest.size()!=0)
	{		spint=spint+integrate(sprest);
	}	sPoisson *aux=new sPoisson(spint);
	return *aux;
}
```